DWBB Datapath Floating-Point Overview

Introduction

Revision History

The DesignWare Building Block (DWBB) floating-point (FP) components constitute functions used to synthesize floating-point computational circuits in high-end ASICs. The functions mainly deal with arithmetic operations in FP format, format conversions, and comparisons. The main features of this library are as follows:

- The format of the FP numbers that determines the precision of the number that it represents is set by a parameter. You can select the precision based on the number of bits in the exponent and significand (or mantissa). The parameters cover all IEEE Std 7544 standard formats.
- Accuracy conforms to the definitions in the IEEE Std 754 standard for the basic arithmetic operations. Improved accuracy is obtained with multi-operand FP components.

Compatibility with the IEEE Std 754 Standards

Using the *ieee_compliance* parameter, all DWBB FP components are configurable to be fully compliant with the IEEE Std 754-1985 standard. In addition, some commonly used components support “not a number” (NaN) representation that is compatible with the IEEE Std 754-2008 and IEEE Std 754-2019 standards.

- When *ieee_compliance* = 0, denormalized numbers are considered zeros and NaNs are considered infinities. This behavior is consistent with the previous Synopsys floating-point components developed for Module Compiler (MC).
- When *ieee_compliance* = 1, the DWBB FP components are compliant with the IEEE Std 754-1985 standard. The NaN value produced sets the least-significant bit (LSB) of the significand portion of the FP output. All NaNs are treated as one type (there is no concept of quiet or signaling types).
- When *ieee_compliance* = 3, NaN representation is compatible with the IEEE Std 754-2008 standard. The NaN value produced sets the most-significant bit (MSB) of the significand portion of the FP output. Both quiet and signaling types of NaNs are supported. In this configuration, denormals support is always enabled. The following components support this level of compliance (*ieee_compliance* = 3):

  - DW_fp_add
  - DW_fp_add_DG
  - DW_fp_addsub
  - DW_fp_addsub_DG
  - DW_fp_div
  - DW_fp_div_DG
  - DW_fp_mult
  - DW_fp_mult_DG
  - DW_fp_sqrt
  - DW_fp_sub
  - DW_fp_sub_DG

For more on how special floating point numbers are represented, see Table 1-5 on page 4.

Formats

The DWBB FP components support more formats than the IEEE Std 754 standards describe. For a given set of parameters, the components use FP formats that directly correspond to the standard (for example, single-precision FP format uses $f = 23$ and $e = 8$).

The FP components support two basic data types: integer and floating point. The length of both data types can be parameterized.

Integer Format

Integers may be signed or unsigned. Unsigned numbers use the standard binary notation. Signed integers use the two's complement notation. The positions of the most-significant bit (MSB) and least-significant bit (LSB) are shown Table 1-1.

Table 1-1 Integer Format Bit Positions

MSB		LSB
n-1	....	0

Floating-Point Format

The FP components use binary floating point representation — the radix of the number is always two. FP numbers are signed-magnitude numbers encoded with three unsigned integer fields: a sign bit, a biased exponent, and a fractional portion of the significand (fraction), as shown in Table 1-2 on page 2.

Table 1-2 Floating Point Number Bit Positions

Sign	Biased Exponent	Fraction Bits of Significand
[e + f]	[e + f-1:f]	[f - 1:0]

As shown in Table 1-2, all the FP formats are defined by two integer values: `e` and `f`, where:

- `e` represents the number of biased exponent bits
- `f` represents the number of fraction bits

The MSB at position (`e` + `f`) holds the sign of the floating-point number. Therefore, a floating-point representation is always `e` + `f` + 1 bits long. This definition is consistent with the IEEE 754 standard.

For convenience, this document maps the fields of a floating-point representation to the following:

- S: sign
- E: biased exponent
- F: fraction

The numerical value of maximum and minimum normalized numbers for a given format are defined as:

$$\text{MinNorm} = (S, 1, 0)$$

$$\text{MaxNorm} = (S, 2^{e} - 2, 2^{f} - 1)$$

Note that the exponent field always contains a biased exponent (E). The value of the bias is defined as $(2^{e-1}-1)$. The value (V) of a normal FP number is calculated as:

$$V = (-1)^S \times (\text{fraction} \times 2^{-f} + 1) \times 2^{E - \text{bias}}$$

Floating Point Format Examples

Formats defined by the IEEE Std 754 standard are widely used, and a custom FP format can also be defined. Examples are shown in the following sections.

Single-Precision Floating Point Format

The IEEE Std 754 single-precision format consists of 32-bit operands. The MSB is the sign bit, with 8 bits of exponent and 23 bits of fraction, as shown in Table 1-3.

Table 1-3 IEEE Std 754 Standard Single-Precision Floating-Point Format

Sign	8-Bit Biased Exponent	23-Bit Fraction Bits of Significand
[31]	[30: 23]	[22:0]

	Zero Exponent = 0	Normalized Value = $(-1)^S$ x (1 + fraction * 2^{-23}) x 2^{E-127}
	Infinity Exponent = 255	Min Norm = $(-1)^S$ x (1.0) x 2^{-126}
	Normal FP Exponent = integer range [1, 254]	Max Norm = $(-1)^S$ x (2 - 2^{-23}) x 2^{127}
	Bias = 127	

Customized 18-Bit Floating Point Format

A custom 18-bit FP format has 6 bits of exponent and 11 bits of normalized FP fraction. The MSB is reserved for the sign bit. Specific format values are shown in Table 1-4 on page 3.

Table 1-4 Customized 18-Bit Floating Point

Sign	6-Bit Biased Exponent	11-Bit Fraction Bits of Significand
[17]	[16: 11]	[10: 0]

	Zero Exponent = 0	Normalized Value = $(-1)^S$ x (1 + fraction * 2^{-11}) x 2^{E-31}
	Infinity Exponent = 63	Min Norm = $(-1)^S$ x (1.0) x 2^{-30}
	Normal FP Exponent = integer range [1, 62]	Max Norm = $(-1)^S$ x (2 - 2^{-11}) x 2^{31}
	Bias = 31	

Special Floating Point Numbers

Special values in the FP system are represented as shown in Table 1-5 on page 4. Important values for exponents are:

- Bias value: $bias = 2^{e-1} - 1$
- Biased Einf (exponent for infinity): $Einf = 2^{e} - 1$
- Maximum biased exponent for a normal number representation: $Emax = Einf - 1$
- Minimum biased exponent for a normal number representation: $Emin = 1$

A normal FP value is represented with a biased exponent E in the range [$Emin,Emax$] and any combination of bits as fraction bits of the significand.

Although the NaN generated by a FP component (when $ieee\_compliance \neq 0$) has a fixed fraction value, the input received by any component is considered a NaN when it has a non-zero fraction value and an exponent value equal to *Einf*.

Table 1-5 Special Floating Point Numbers

Floating Point Number	FP Representation (Sign, Biased Exponent, Fraction)
+/- Zero	(0 or 1, 0, 0)
+/- Infinity	(0 or 1, Einf, 0)
NaN*	(0, Einf, ≠ 0) (when generated by a floating point component) (0 or 1, Einf, ≠ 0) (as input)
Denormalized value (denormal)*	(0 or 1, 0, any bit vector)

* These special numbers are used when *ieee_compliance* is not set to 0. The value of a denormalized FP number is given as $(-1)^{S} \times (fraction \times 2^{-f}) \times 2^{1 - bias}$.

Rounding

Rounding is the process by which a number, regarded as infinitely precise (unbounded exponent and infinitely precise significand), is mapped to a limited precision FP format. All the basic arithmetic functions in the Floating-Point library behave as though they first produce an intermediate result correct to infinite precision, and then round this intermediate result to fit the FP format. Special conditions apply to the more complex components and, when applicable, are described in the component datasheet.

For integers, rounding implies generating an output that has no data to the right of the binary point.

For FP numbers, rounding means that the significand is reduced to *f* bits of precision to the right of the binary point.

In both cases, the long precision results are first truncated (the least-significant bits (LSBs) are discarded) to get the desired number of bits. This significand is part of one FP number that approximates the infinite precision result. There is another FP number immediately larger (in magnitude) that is obtained by incrementing the truncated significand and making adjustments to have the significand normalized. So, the rounding method defines how a decision is made to use one of these two numbers as a valid approximation for the infinite precision result. Note that the sign of the result is also considered in the rounding procedure.

The exponent values used in the representations of FP numbers at the significand rounding phase are not bounded, which means the rounding procedure is focused on the value of the significand only. It also means that the approximated numbers may be outside the range of representable numbers, which is considered an exception and must be treated separately.

Note that the material above describes functional behavior, not necessarily hardware implementation.

Also, some components do not have rounding control. In these cases, the component uses a single method to keep the rounding error inside bounds, and is documented in the corresponding datasheet.

Rounding FP Significands

The rounding modes determine the conditions under which the internal long-precision result's significand is reduced to fit the precision defined for the target FP format. The FP components support dynamic selection of rounding modes, which is determined by a 3-bit input signal named `rnd`.

Table 1-6 describes the rounding modes in terms of the near floating-point values *F1* and *F2* (*F1* < *F2*) of an infinitely precise F.

Table 1-6 Rounding Modes

rnd	Rounding Mode	Alias	Description
3'b000	IEEE round to nearest even	even	Round to the nearest FP number If *F1* and *F2* are equally near, choose the one with the even significand (the one with LSB = 0).
3'b001	IEEE round to zero	zero	Use *F1* if (F ≥ 0) or *F2* if (F < 0)
3'b010	IEEE round to positive infinity	+ ∞	Round to *F2*
3'b011	IEEE round to negative infinity	- ∞	Round to *F1*
3'b100	Round to nearest up	up	Round to the nearest FP number If *F1* and *F2* are equally near, then use *F2* if (F ≥ 0) or *F1* if (F < 0). NOTE: This rounding mode is valid only when *ieee_compliance* ≠ 3. For components that support *ieee_compliance* = 3, this value of `rnd` is mapped to 3'b000 (IEEE round to nearest even).
3'b101	Round away from zero	away	Use *F2* if (F ≥ 0) or *F1* if (F < 0) NOTE: This rounding mode is valid only when *ieee_compliance* ≠ 3. For components that support *ieee_compliance* = 3, this value of `rnd` is mapped to 3'b0001 (IEEE round to zero).
3'b110	Reserved		
3'b111	Reserved		

In all rounding modes, if the LSBs being discarded from the infinitely precise significand of F are zeros, then *F1* or *F2* matches F exactly and there is no rounding. When the LSBs being discarded are non-zero, the infinitely precise significand lies strictly between two representable significands, and the rounded number is, therefore, inexact. The rounded floating-point number (*F_rnd*) is decided to be *F1* or *F2*, based on the rounding mode and the value/sign of F. It may be the case that the exponent of *F_rnd* is too large (overflow) or too small (underflow) to be represented as a FP value in the given format. Those cases are called exceptions and are described in "Exceptions" on page 6.

Independent of the value of parameter *ieee_compliance*, all internal calculations of FP components handle denormalized values, and therefore F may be a denormalized FP value.

When *ieee_compliance* ≠ 0, *F1* and *F2* can be identified for a denormalized value F, and the rules listed in Table 1-6 are used to generate the final output.

When *ieee_compliance* = 0, *F_rnd* may need to be adjusted, as described in "Exceptions" on page 6.

For some algorithms used to implement FP operations, it is not possible to control the rounding direction. In these cases, the FP component may deliver either *F1* or *F2* as the result of a given computation. The actual value delivered depends on the inputs only, and not on the rounding control. This is called *faithful rounding* in the datasheets and related documentation.

Exceptions

For integers, an exception happens when the rounded significand exceeds the range of the output integer.

For floating-point formats, an exception occurs when FP significand rounding yields a number *F_rnd* (with an unbounded exponent) that is "out-of- bounds". Out-of-bounds means that the rounded floating-point value has a biased exponent that is outside the range of normal floating-point values for the given format.

- Overflow happens when the absolute value of the rounded floating-point value is greater than MaxNorm.
- Underflow happens when the absolute value of the rounded floating-point is less than MinNorm and it is not an exact zero.

More precisely, these conditions can be defined as a function of the biased exponent of *F_rnd* (call it e_rnd), as follows:

- Overflow: e_rnd > Emax
- Underflow: e_rnd < Emin = 1

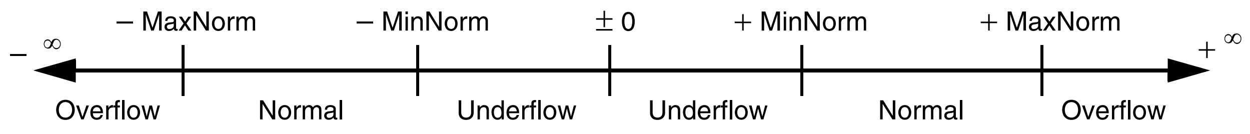
These exceptions are indicated in the status flags as HugeInt, Huge, and Tiny (for more about status flags, see "Status Flags" on page 7). When an exception occurs, the FP components respond as follows:

- For integers, if the rounded result exceeds the largest representable integer with the same sign (HugeInt), the largest representable integer with the correct sign will be the output, regardless of the rounding mode.
- For floating-point values:
  - When not using denormal numbers, if overflow or underflow is detected, the output is determined based on the rounding mode and the sign of the infinitely precise result, as described in Table 1-7.
  - When using denormals, the significand is denormalized (shifted to the right) until the exponent is 1. Rounding is done to the denormalized significand to keep only *f* bits.

Table 1-7 Exception Handling After Rounding with Unbounded Exponent

		Rounding Mode					
Condition	**Sign**	**even**	**zero**	**+ ∞**	**– ∞**	**up**	**away**
Overflow	+	+ ∞	+ MaxNorm	+ ∞	+ MaxNorm	+ ∞	+ ∞
	–	– ∞	– MaxNorm	– MaxNorm	– ∞	– ∞	– ∞
Underflow (when not using denormals)	+	+ 0	+ 0	+ MinNorm	+ 0	+ 0	+ MinNorm
	–	– 0	– 0	– 0	– MinNorm	– 0	– MinNorm

The number line below shows the key representable floating-point numbers across the top, and the various ranges of results across the bottom. This illustration provides a useful view of the representable numbers and an intuitive basis for Table 1-7.



Status Flags

Every FP component in the library has an output port called `status`. The component generates a number of flags that depend on the result. Each bit in the `status` output indicates a different condition as detailed in Table 1-8.

Table 1-8 Status Flags

Bit	Flag	Description
0	Zero	The integer or floating point output is zero.
1	Infinity	The floating point output is infinity.
2	Invalid	The floating point operation is not valid[a]. Also, if *ieee_compliance* = 1 or 3, this flag is set when one or both of the inputs is NaN.
3	Tiny	A rounded floating-point number with unbounded exponent has a magnitude less than the minimum normalized number, and it is not an *exact zero*[b].
4	Huge	A rounded floating-point number with unbounded exponent has a magnitude greater than the maximum normalized number.
5	Inexact	An integer or floating point output is not equal to the infinitely precise result.
6	HugeInt	An integer result after rounding has a magnitude greater than the largest representable two's complement integer with the same sign.

Table 1-8 Status Flags (Continued)

Bit	Flag	Description
7	CompSpecific	This flag has its meaning specified for some DW components and is noted in component datasheets. When not described in a component datasheet, this flag is not used and has a value of 0.

a. Invalid operations that set the INVALID status flag are defined in the IEEE 754 standard, such as:
- Any operation on a NaN input (ieee_compliance = 1)
- Addition of infinities with opposite signs, or subtraction of infinities with same sign.
- Multiplication of zero and infinity.
- Division 0/0 and infinity/infinity.
- Square root of a negative FP value.

b. When a floating point value, calculated with infinite precision (without rounding), is a zero, then the value is an “exact zero.”

Special Operations

There are some cases when the result of an FP operation is not obvious or is considered invalid:

- Exact zero differences — when the operation is effective subtraction and the result is exactly zero, the sign of the result is determined by the rounding mode. If the rounding mode is *RND* = 3, the output is -0. In all other rounding modes, the output is +0.
- Comparison of zeros — the sign of zeros is not used for comparison (+0 = -0).
- Integer overflows — if HugeInt flag is set, the largest representable integer with correct sign is output. For positive numbers, the output is 2^{n-1} - 1, and for negative numbers the output is -2^{n-1}.
- Subtraction of infinities — in this case the Invalid flag is asserted and the output is set to NaN (when *ieee_compliance* ≠ 0) or infinity (when *ieee_compliance* = 0).
- Product of Zero and Infinity — the Invalid flag is asserted and the output is set to NaN (when *ieee_compliance* ≠ 0) or infinity (when *ieee_compliance* = 0).
- Complex FP operations, such as dot-products, may assert the Invalid flag based on internal invalid operations. For example, the internal products generated by the component are infinities with opposite signs, forcing the rule related to the subtraction of infinities to be applied. More information about these situations are given in the individual component datasheets.

Revision History

For notes about this release, see the *DesignWare Building Block IP Release Notes*.

For a version of this datasheet with visible change bars, click here.

Date	Release	Updates
October 2020	DWBB_202009.1	■ Added details for *ieee_compliance* = 3 and enhanced support for "not a number" (NaN) behavior for some floating point components ■ Adjusted organization of some information
April 2020	DWBB_201912.3	■ In "Rounding FP Significands" on page 5: - Adopted the new title; previously, it was "Rounding" - Clarified some language and concepts - Removed the table "Rounding Denormalized Values when ieee_compliance = 0" ■ Clarified some language and concepts in "Exceptions" on page 6
April 2019	DWBB_201903.1	■ Clarified IEEE 754 compatibility in "Floating Point Format Examples" on page 3 ■ Added this revision history table and the document links on this page